TEMPORAL LOGIC AS A SIMULATION LANGUAGE

by

Alexander Tuzhilin
Information Systems Department
Leonard N. Stern School of Business
New York University
40 West 4th Street, Room 624
New York, New York 10003

October, 1990

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-90-22

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

Temporal Logic as a Simulation Language

Alexander Tuzhilin

Information Systems Department
Stern School of Business
New York University
40 West 4th Street, Room 624
New York, NY 10003
phone: 212-998-4203
e-mail: tuzhilin@rnd.gba.nyu.edu

Abstract

We advocate the use of temporal logic instead of the first-order logic in rules of knowledge-based simulation systems. We argue that this provides several advantages that will be discussed in the paper. We show how temporal logic is used in simulation by considering language PTL based on temporal logic programming.

1 Introduction

Recently, there has been substantial interest developed in knowledge-based simulation methods. Books by Elzas, Ören and Zeigler [EÖZ86, EÖZ89] and volumes 17 and 18 of the Simulation Series [LA86, LB87] contain many articles on the subject. Also, Rothenberg [Rot89] presents a recent tutorial of the area. Many knowledge-based simulation systems provide support for rule-based and object-oriented paradigms combined with a support for powerful knowledge representation schemes such as frames. Examples of commercial systems of this type are SIMKIT [Int85] and Simulation Craft [SFBB86].

The rule-based component of these systems is typically based on a logic programming language, e.g. PROLOG, or on a production system, e.g. OPS5 [BFK86]. Therefore, rules used in the knowledge-based simulation methods described above are based on the first-

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

order logic since logic programming languages and production systems have their roots in the first-order logic.

Since simulation methods deal with processes evolving in time and since the first-order logic does not support time directly, the knowledge-based simulation methods must provide an explicit support for time. For example, most of the methods explicitly define and manipulate a system clock and provide some form of event scheduling.

In this paper, we propose to use a predicate *temporal logic* instead of the first-order logic in the rule-based simulation methods. To that extent, we consider a language PTL [Tuz90] constituting a rule-based fragment of *temporal logic* and explain how PTL can be viewed as a simulation language. Since temporal logic is a logic of time, we will see that there is no need to define time explicitly as a parameter in rules in the simulation methods based on temporal logic. In particular, there is no need to maintain a system clock and schedule events in PTL since both of these tasks become the implementation issues of the language and should not concern the user of PTL. Better support for time and a higher level of declarativeness constitute only one advantage of simulation methods based on temporal logic over the methods based on the first order logic. Other important advantages will be presented in Section 3.

In this paper, we mainly advocate the use of temporal logic in knowledge-based simulation systems. To make a real simulation system based on temporal logic, it is important to extend PTL with additional features. For example, it is important to provide a support for object-oriented programming, develop a good user friendly syntax, possibly, based on natural language, a good user interface, and provide a support for advanced knowledge-representation schemes, e.g. frames. Such extensions will further increase modeling capabilities of the language. However, the treatment of these issues is beyond the scope of this paper and constitutes a topic of current research.

To make the paper self-contained, we briefly review PTL [Tuz90] in the next section.

2 PTL as a Simulation Language

In order to define PTL, we introduce the following preliminary concepts from temporal logic. The books by Kroger [Kro87] and Rescher and Urquhart [RU71] provide a good

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

introduction to temporal logic. First, we use standard temporal operators of *necessity* ($\Box$) and *past necessity* ($\blacksquare$), *possibility* ($\Diamond$) and *past possibility* ($\star$), *next* ($\circ$) and *previous* ($\odot$) [Kro87, RU71]. Since predicates are defined over time in temporal logic, it means that the temporal operators generate predicates that are also defined over time. For example, $\Box A$ is true at time t if A is true at time t and at *all* times after time t; $\Diamond A$ is true at time t if A is true either at time t or at *some* time after t; and $\circ A$ is true at time t if A is true at time $t+1$. Similarly, we can define past "mirror" images of these operators. Second, we introduce two new temporal operators of *bounded necessity* $\Box_T$ and *bounded possibility* $\Diamond_T$ together with their past mirror images. $\Box_T A$ is true at time t if A is true from time t up to but not including $t+T$ and is false at time $t+T$. $\Diamond_T A$ is true at time t if there is t', such that $t \leq t' \leq t+T$ and A is true at t'. Examples of various temporal operators will be provided in Example 1.

A *PTL program* is a set of temporal clauses. A temporal clause has the form $BODY \rightarrow HEAD$, where $BODY$ is *any* temporal logic formula with only the *past* temporal operators appearing in them; and $HEAD$ is a conjunction of next-literals, necessity and bounded necessity operators and their negations. As Example 1 shows, conjunctions in the head of a rule will be denoted with semicolon (;). It follows from this definition that the body of a rule refers to the current moment of time and to the past, whereas the head of a rule refers strictly to the future. In addition, PTL supports negations both in the head and the body of a rule.

Next, we provide examples of PTL rules, as presented in [Tuz90], and explain various points about PTL using this example.

Example 1 A Flexible Manufacturing System (FMS) manufactures certain products such as car engines, electronic boards, or electrical appliances. In this example, we assume that an FMS performs only assembly operations on unfinished units. The initial part of an assembly is brought into the system through the *load-unload* station. Then it is carried among various manufacturing units, called *cells*, where the assembly process takes place. For example, in case an FMS manufactures toasters, one cell can be responsible for making the outer body of a toaster, another for installing heating elements in it, another for assemblying knobs on the front panel of a toaster, and still another one for attaching the front door to it. A special vehicle, called an *Automatic Guidance Vehicle (AGV)*, carries incomplete assemblies among

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

various cells. When the assembly process is completed, the finished units are brought by AGVs back to the load-unload station where they are removed from the FMS system.

The state of such an FMS system is defined with the following predicates. $D(AGV, C)$: a vehicle AGV is docked at a cell C; $L1(ASM, AGV)$: an assembly ASM is loaded on a vehicle AGV; $L2(ASM, C)$: an assembly ASM is located in a cell C; $MOV(AGV, C)$: an AGV AGV is moving to cell C; $NEXT(C, C')$: the next assembly operation is done in cell C' after the previous assembly operation is done in cell C; $PROC(C, T)$: it takes T units of time to perform an operation in cell C; $TRAV(C, C', T)$: it takes T units of time for an AGV to travel from cell C to cell C'. Predicates $NEXT$, $PROC$ and $TRAV$ are *rigid* [AM89]: they don't change over time.

Examples of several PTL rules partially describing behavior of an FMS are presented now. We assume that a cell never processes the same assembly twice.

R1: If an AGV is docked at a cell with an assembly on it and the assembly has been processed by the cell (in the past), then move the AGV to the next cell for the time period determined by relation $TRAV$.

$$D(AGV, C) \wedge L1(ASM, AGV) \wedge \star L2(ASM, C) \wedge NEXT(C, C') \wedge TRAV(C, C', T) \rightarrow \circ\neg D(AGV, C); \Box_T MOV(AGV, C')$$

R2: If an AGV arrived at a cell[1] then dock it at that cell.

$$\odot MOV(AGV, C) \wedge \neg MOV(AGV, C) \rightarrow \circ D(AGV, C)$$

R3: If an AGV is docked at a cell with an assembly loaded on it that has not been processed by the cell yet, and no other assembly is in that cell, then transfer the assembly from the AGV into the cell. Let it stay in the cell for the time period determined by relation PROC.

$$D(AGV, C) \wedge L1(ASM, AGV) \wedge \neg \star L2(ASM, C) \wedge (\forall ASM')\neg L2(ASM', C) \wedge PROC(C, T) \rightarrow \circ\neg L1(ASM, AGV); \Box_T L2(ASM, C)$$

R4: If an operation on an assembly is finished by the cell and an empty AGV is docked at the cell then put the assembly on the AGV.

[1] In other words, it was moving in the previous time moment and stopped moving at present.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

$$\odot L2(ASM, C) \wedge \neg L2(ASM, C) \wedge D(AGV, C) \wedge (\forall ASM')\neg L1(ASM', AGV)$$
$$\rightarrow \circ L1(ASM, AGV)$$

Note the usage of temporal operators *past possibility* ($\star$) and *future bounded necessity* ($\Box_T$) in rules **R1** and **R3**. $\star L2(ASM, C)$ means that the assembly ASM was located in the cell C at some time in the past; $\Box_T MOV(AGV, C)$ means that the vehicle AGV is moved to the cell C for T time units. Also note the usage of negations ($\neg$) both in heads and bodies of rules.

■

The meaning of a PTL program is associated, as in the case of a logic program, with a certain *model* of that program [Tuz90]. In case of temporal logic, a model of a program is defined in terms of the instances of program predicates taken at *all* the moments of time; in addition, the rules in the program must be true at *all* the moments of time as well. In other words, a model of a PTL program can be considered as a *trajectory* of all the program predicates over time. If program predicates represent a state of a system then the trajectory of its values can be viewed as a *simulation trace* and the PTL program as a simulation program producing the trace.

A PTL program can have many models in general. In order to be able to select a unique model of a PTL program, [Tuz90] introduces *inflationary* and *boundary conditions*. These two conditions reduce the class of all the models of a PTL program to a unique model which becomes *the* meaning of the program.

To summarize, a PTL program has a unique model (determined by inflationary and boundary conditions) and this model can be interpreted as a simulation trace of the program.

Notice that we did not provide any way to *compute* the simulation trace. We simply stated that such a trace *exists* for any PTL program. Because PTL does not specify how to compute the simulation trace of a program and only defines the trace *in terms of* the program, it provides an example of a *declarative* rule-based simulation language. As will be stated in the conclusion, efficient methods to compute the simulation trace of a PTL program constitute the topic of current research.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

3 Advantages of PTL as a Rule-Based Simulation Languages

In Section 2 we proposed to use rules based on temporal logic as opposed to the first order logic in knowledge-based simulation systems. Usage of temporal logic improves simulation capabilities of knowledge-based systems because of the following reasons.

1. Temporal logic produces more declarative simulation programs. A simulation trace is the *model* of a program. Therefore, there is no need for a user to define any computational mechanisms in a program such as instructions on how to maintain a system clock or an event queue. As Example 1 shows, the simulation trace of the program specified by rules **R1** - **R4** is uniquely defined by these rules and by initial conditions and does not depend on anything else.

2. Temporal logic provides powerful mechanisms to describe state transitions. The body of a rule can depend not only on the current state of the system but also on its past history (by using past bounded and unbounded possibility and necessity operators). Also, the head of the rule can describe not only what happens to the system at the next moment of time but also at more distant future instances. This can be achieved by using temporal operators of necessity, bounded necessity and multiple nexts.

3. In [Tuz90], a systematic and declarative query language for PTL programs was proposed. This language is also based on temporal logic. For example, a query "Find all assemblies that will visit cell C_0 within the next 20 minutes" on a PTL program from Example 1 can be expressed in this query language as

$$\{ASM \mid \diamond_{20} L2(ASM, C_0)\}$$

   This language can also be viewed as a query language on simulation traces.

   Therefore, temporal logic can be used both for simulating behavior of a system and for asking queries about its behavior. This means that temporal logic provides "seamless" integration between the simulation method and the query language about the results of the simulation since both of them are based on temporal logic.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

4. Temporal logic can support *partial simulations.* Traditionally, when a query is asked about a system being simulated, a simulation is run first and then the query is asked on the simulation trace being produced. If the query language referred to in item 3 is combined with PTL then we can ask a query first and then run the simulation *only* on the part of the system pertinent to the query. For example, if we want to know if an assembly leaves a cell within the next 10 minutes, probably, there is no need to simulate the behavior of the entire FMS. It may suffice to simulate the processes within a single cell for 10 minutes thus producing a partial simulation of an FMS for the query asked.

5. The same formalism of temporal logic can be used not only for describing behavior of a system and asking queries about this behavior, but also for specifying *dynamic constraints* on the behavior. A dynamic constraint [Bro81, CPB81, CF84, Via88] imposes restrictions on possible instances of predicates over time. For example, a dynamic constraint can state that the salary of an employee cannot decrease over time. As [CF84] advocates, temporal logic is well-suited for the specification of dynamic constraints.

6. Temporal logic constitutes a solid and well-studied formalism. Therefore, rules based on temporal logic have a solid theoretical foundation.

7. PTL can be reduced to the first-order logic and to production systems [BFK86] in the degenerate case when no temporal operators are used in the body of a rule and only *next* ($\circ$) temporal operators are used in the head [Tuz90]. Therefore, PTL is compatible with these two formalisms and inherits their strong properties.

For all the reasons listed above, temporal logic improves simulation capabilities of knowledge-based systems and, therefore, can be used instead of the first-order logic in these systems.

4 Related Work

There has been much work done on using rule-based systems in simulation. Books by Elzas, Ören and Zeigler [EÖZ86, EÖZ89] and volumes 17 and 18 of the Simulation Series

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

[LA86, LB87] contain many articles on the subject. However, none of these rule-based systems use temporal logic.

The use of temporal logic to model behavior of Computer Integrated Manufacturing Systems was suggested by Chaudhury and Rao [CR88] and by Huber and Buenz [HD89]. Huber and Buenz used Allen's interval-based temporal logic to specify constraints on production schedules; however, they have not used it as a method to generate computations. Chaudhury and Rao [CR88] suggested the possibility of using temporal logic for the conceptual modeling of CIM systems.

As was stated before, the language PTL was introduced in [Tuz90]. This language is related to the work on temporal logic programming [AM89, Bau89, Gab89, KKN+90, FKTMo86, Mos86] and constitutes an extension of this work to support requirements for modeling reactive systems. See [Tuz90] for more details on comparison of PTL with the referenced work on temporal logic programming.

5 Conclusion and Work in Progress

In the paper we argue that temporal logic is well-suited for knowledge-based simulations. It provides a highly declarative method of modeling behavior, has powerful mechanism to describe state transitions, supports partial simulations, can be used for the specification of query languages and dynamic integrity constraints, and provides a sound theoretical basis for knowledge-based simulation systems.

We also considered language PTL [Tuz90] and showed how it can be used to simulate behavior of an FMS.

We are currently working on two issues. First, we are trying to extend PTL to support object-oriented programming and advanced knowledge representation methods. Second, we are trying to find efficient methods to compute simulation traces of PTL programs.

References

[AM89] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

[Bau89] M. Baudinet. Temporal logic programming is complete and expressive. In *Symp. on Principles of Programming Languages*, pages 267–280, 1989.

[BFK86] L. Brownston, R. Farrell, and E. Kant. *Programming Expert Systems in OPS5: an Introduction to Rule-Based Programming.* Addison-Wesley, 1986.

[Bro81] M. Brodie. On modelling behavioral semantics of databases. In *International Conference on Very Large Databases*, pages 32–42, 1981.

[CF84] Marco A Casanova and Antonio L. Furtado. On the description of database transition constraints using temporal languages. In *Advances in Database Theory*, pages 211–236. Plenum Press, 1984. vol. 2.

[CPB81] S. Ceri, G. Pelagatti, and G. Bracchi. Structured methodology for designing static and dynamic aspects of database applications. *Information Systems*, 6:31–45, 1981.

[CR88] A. Chaudhury and H. R. Rao. Conceptual modeling in computer integrated manufacturing systems. In M.D. Oliff, editor, *Expert Systems and Intelligent Manufacturing*, pages 265–276. Elsevier Science Publishing Co., 1988.

[EÖZ86] M.S. Elzas, T.I. Ören, and B.P. Zeigler, editors. *Modelling and Simulation Methodology in the Artificial Intelligence Era.* North-Holland, 1986.

[EÖZ89] M.S. Elzas, T.I. Ören, and B.P. Zeigler, editors. *Modelling and Simulation Methodology: Knowledge Systems' Paradigms.* North-Holland, 1989.

[FKTMo86] M. Fujita, S. Kono, H. Tanaka, and T. Moto-oka. Tokio: Logic programming language based on temporal logic and its compilation to Prolog. In *Third International Conference on Logic Programming*, pages 695–709. Springer-Verlag, 1986. LNCS 225.

[Gab89] D Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Colloquium on Temporal Logic in Specification*, pages 402–450. Springer-Verlag, 1989. LNCS 398.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

[HD89] A. Huber and Buenz D. Using GRAI to specify expert systems for the control and the supervision of flexible flow lines. In J. Browne, editor, *Knowledge Based Production Management Systems*, pages 295–308. Elsevier Science Publishers, 1989.

[Int85] IntelliCorp, Mountain View, Calif. *The SIMKIT System: Knowledge-Based Simulation Tools in KEE*, 1985.

[KKN+90] D. Kato, T. Kikuchi, R. Nakajima, J. Sawada, and H. Tsuiki. Modal logic programming. In *VDM and Z - Formal Methods in Software Development.* Springer-Verlag, 1990. LNCS 428.

[Kro87] Fred Kroger. *Temporal Logic of Programs.* Springer-Verlag, 1987. EATCS Monographs on Theoretical Computer Science.

[LA86] P. Luker and H.H. Adelsberger, editors. *Intelligent Simulation Environments*, volume 17. SCS Simulation Series, 1986.

[LB87] P. Luker and G. Birtwistle, editors. *Simulation and Artificial Intelligence*, volume 18. SCS Simulation Series, 1987.

[Mos86] B. Moszkowski. *Executing Temporal Logic Programs.* Cambridge University Press, Cambridge, England, 1986.

[Rot89] J. Rothenberg. Tutorial: Artificial intelligence and simulation. In E.A. MacNair, K.J. Musselman, and P. Heidelberger, editors, *Proceedings of the SCS Winter Simulation Conference*, 1989.

[RU71] Nicholas Rescher and Alasdair Urquhart. *Temporal Logic.* Springer-Verlag, 1971.

[SFBB86] N. Sathi, M. Fox, V. Baskaran, and J. Bouer. Simulation Craft: An artificial intelligence approach to the simulation life cycle. In *Proceedings of the SCS Summer Simulation Conference*, 1986.

[Tuz90] A. Tuzhilin. Programming reactive systems in temporal logic. Working Paper IS-90-21, Stern School of Business, NYU, 1990.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22

[Via88] V. Vianu. Database survivability under dynamic constraints. *Acta Informatica*, 25:55–84, 1988.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-22